D1 TODO 1: Problem Domain & Knowledge Domain D

Problem Domain

Access control in software systems.

Knowledge Domain D

Role-Based Access Control (RBAC) access decision validation.

More precisely:

> The knowledge domain focuses on determining whether a user should be granted or denied access to a protected resource in a multi-user software system, based on assigned roles, associated permissions, and access constraints.

This domain includes experiential knowledge related to:

- Correct interpretation of roles and permissions
- Resolution of access requests involving multiple roles
- Enforcement of access constraints (e.g., ownership, resource state, or contextual conditions)
- Prevention of unauthorized access resulting from misconfiguration or overly permissive role assignments

RBAC is widely used in real-world systems, and authoritative public information is available through standards, technical documentation, and academic literature. Practical experience in designing and validating RBAC policies can be beneficial to developers and system administrators, as access control errors are a common source of security vulnerabilities.

D1 TODO 2: Goal G (SMART)

Goal G

> The goal of this project is to develop an expert system that assists in validating access control decisions in a role-based access control (RBAC) system by determining whether a specific access request should be granted or denied, based on defined roles, permissions, and constraints.

SMART justification:

- **Specific**: Focuses on access decision validation (grant vs deny)
- **Measurable**: Produces a clear decision with supporting explanation
- **Achievable**: Based on explicit facts and rules
- **Relevant**: Addresses a common and critical software security problem
- **Time/Scope bounded**: Limited to RBAC reasoning, not full system implementation

---

D1 TODO 3: Potential user U

User U

> The intended user of the expert system is a junior software developer or system administrator responsible for implementing or maintaining access control policies in a software system.

User characteristics:

- Has basic knowledge of users, roles, and permissions
- May lack deep security expertise
- Needs assistance verifying whether access rules are correctly enforced
- Benefits from explainable decisions that justify why access is granted or denied

The expert system supports the user by providing consistent, rule-based access decisions and by reducing errors caused by incorrect assumptions or incomplete access logic.

D1 TODO 4: Knowledge Base K = F ∪ R

Factbase F

Fact categories (design-level)

Facts fall into these groups:

- User facts
- Role facts
- Permission facts
- Resource facts

- Access request facts
- Context / constraint facts

Structured English — Fact definitions

15 useful facts:

1. A user has an identifier.
2. A user is assigned one or more roles.
3. A role grants one or more permissions.
4. A permission allows an action on a resource type.
5. A resource has a type.
6. A resource has an owner.
7. An access request specifies a user, an action, and a resource.
8. A role may be marked as privileged.
9. A role may inherit permissions from another role.
10. A user may have multiple roles simultaneously.
11. A resource may be in a specific state (e.g., active).
12. Some actions are restricted to resource owners.
13. Some permissions require privileged roles.
14. Access decisions may depend on contextual constraints.
15. An access decision is either granted or denied.

facts.clp found in GitHub repository.

Rulebase R

Deny-by-default model (security best practice).

Rules grouped logically:

1. Basic authorization rules
2. Role-permission matching rules
3. Ownership constraint rules
4. Privilege constraint rules
5. Conflict & default rules

20 Structured English Rules:

Basic rules

1. If a user is not assigned any role, deny access.
2. If a requested action is not permitted by any role of the user, deny access.
3. If a user has a role that permits the requested action on the resource type, access may be granted.

Ownership rules

4. If an action requires ownership and the user is not the resource owner, deny access.
5. If the user is the resource owner and the role allows the action, grant access.

Privilege rules

6. If an action requires a privileged role and the user does not have one, deny access.
7. If the user has a privileged role that permits the action, grant access.

Resource state rules

8. If a resource is archived, deny modification actions.
9. If a resource is active, allow modification actions if permitted by role.

Multi-role reasoning

10. If any assigned role permits the action, access may be granted.
11. If roles conflict, deny access.

Safety rules

12. If critical facts are missing, deny access.
13. If the access request is malformed, deny access.

Default rules

14. If no rule explicitly grants access, deny access.

Explanation rules

15. If access is denied due to missing permission, record explanation.
16. If access is denied due to ownership constraint, record explanation.

Consistency rules

17. If a role is privileged, it must have at least one permission.
18. If a permission refers to an unknown resource type, deny access.

Audit rules

19. If access is granted, log the granting rule.
20. If access is denied, log the denying rule.

rules.clp found in GitHub repository.

Debugging & explainability

Debugging is supported by “decision” facts storing reason strings, and the use of CLIPS tracing.

Validation & verification

The knowledge base was validated by testing multiple access scenarios, including valid access requests, unauthorized requests, ownership violations, and missing-role and resources cases. Asymmetric testing was performed by intentionally modifying or removing scenarios.

test.clp found in GitHub repository.